An Adaptive Higher Order Scheduling Policy With An Application To Biosignal Processing

Georgios Drakopoulos

MDAKM Lab
Computer Engineering and Informatics Department
University of Patras, Patras 26504, Hellas
e-mail: drakop@ceid.upatras.gr

Vasileios Megalooikonomou

MDAKM Lab and DB Lab
Computer Engineering and Informatics Department
University of Patras, Patras 26504, Hellas
e-mail: vasilis@ceid.upatras.gr

***Abstract*—Efficient biosignal processing requires a solid algorithmic background as well as lightweight infrastructure. Job scheduling, namely the policy determining the maximum process number and the order of the actual process execution from the CPU, is closely related not only to performance but also to quality of service (QoS). Thus, scheduling policies are important infrastructure components along with hardware, network protocols, and memory management algorithms to name a few. In a typical workstation setting scheduling is local and constitutes part of the operating system, whereas in distributed systems such as Hadoop there is an additional global scheduling layer at the cluster level. An adaptive scheduling algorithm at the operating system level suitable for CPU bound processes and based on estimating higher order moments of the process size distribution is proposed. Then it is compared to FIFO, SJF, and RR policies with synthetic data derived from the standard MIT-BIH electrocardiogram (ECG) dataset serving as performance and quality benchmark.**

Index Terms*—Biosignal processing; Cardiovascular time series; Scheduling policy; Higher order moments; Poisson distribution; Binomial distribution; Support analytics; *autotools

I. Introduction

The design and deployment of successful bioengineering tools rely heavily on both computational and algorithmic frameworks because of their sensitive, human-centric nature as well as because of their high data intensity. In many realistic scenaria, such as frailty assessment, where a heterogenous subject group is under continuous study or monitor a relatively smooth result flow is desired even for offline analytics in order for the healthcare assistants or medical personnel to be fully utilized as well as for population anomalies to be timely discovered. This adds a new quality of service (QoS) criterion for evaluating bioengineering systems besides the traditional performance-oriented ones.

One way for achieving both quality and performance obectives is adding flexible job scheduling policies such that system workload remains balanced under logical assumptions and the user perception of the mean turnaround time remains satisfactory. Policy adjustment is one of the least invasive methods to alter performance and quality dynamics and, consequently, many systems offer APIs for dynamically doing so. This principle applies to both workstation and distributed settings alike. In fact, in infrastructure falling under the latter category there are at lest two independent policy, one for local node scheduling and one for global job control. Depending on system configuration, local and global policies might interact, although in practice for resiliency, abstraction, and scaling purposes they are completely independent [1].

In order to better meet the above requirements, scheduling policies gradually came to rely on a number of methods ranging from linear or maximum likelihood estimators and neural networks to time series predictors and association rules. This is similar, albeit more sophisticated, to the branch prediction techniques employed by advanced hardware since at least the 1980s [2][3]. That has by no means led to the extinction of traditional policies which are agnostic to the process number or to their individual or statistic characteristics which are simpler and, hence, computationally lighter and easier to implement.

The primary contribution of this paper is a flexible scheduling policy based on higher order moments of the estimated process size distribution. The proposed policy along with the widespread policies of *first-in first-out* (FIFO), *round robin* (RR), and *linear estimation* (LE) have been implemented in C for a typical Linux workstation. Under every of these four scheduling policies have been executed two distinct process groups. Each of these groups performs an offline bioengineering computation, namely the Z-score and the Görtzel algorithm. The standard MIT-BIH dataset has been used to generate the benchmark data for evaluating both the performance and the perceived QoS of the four abovementioned scheduling policies. It should be highlighted that a conscious effort was placed on keeping production tools to a minumim. Thus, the simulator and the auxiliary programs were written in C, developed and maintained with standard *autotools*.

The remaining of this work is structured as follows. Section II briefly reviews the scientific literature regarding scheduling policies and bioengineering analytics. Higher order statistics are discussed in section III. The proposed scheduling algorithm is outlined in section IV along with the performance and quality criteria it is designed to meet. Sections V and VI described the datasets and the results obtained from executing the four scheduling policies for processes computing the Z-score and

for the Göertzel algorithm respectively, whereas section VII summarizes the findings and outlines certain future research directions. Table I summarizes the notation of this paper. Data are represented as real column vectors unless explicitly stated otherwise in boldface small letters and are always indexed from zero for compatibility with the C implementation and, for the Görtzel algorithm, for the Fourier frequency components. Acronyms are defined the first time they appear in text. Finally, the terms *process* and *job* are used interchangeably throughout the text. The same holds about job *size* and *length*.

TABLE I
SYMBOLS USED IN THIS PAPER.

Symbol	Meaning
$\triangleq$	Definition or equality by definition
$\mathrm{E}[X]$	Mean value of random variable (r.v.) X
$\mathrm{E}[X^p]$	p-th noncentral moment of r.v. X
$\mathrm{Var}[X]$	Variance of random variable X
$M_X(s)$	Moment generating function of r.v. X
$\hat{\mu}_0$	Data sample mean (data implied)
$\hat{\sigma}_0$	Data sample variance (data implied)
$J(x)$	Length in terms of time steps of process x
ρ_0	Process service rate or workload, $0 \leq \rho_0 < 1$
$\{s_k\}$	Set consisting of elements s_k
$\lvert S \rvert$	Cardinality of set S

II. Related Work

Analytics form the algorithmic cornerstones of bioengineering. Currently there is a vast array of processing algorithms for one dimensional biosignals such as cardiovascular and respiratory time series, two dimensional such as biomedical images, and multidimensional like blood biochemical examinations. In [4] an iterative regularization algorithm based on finite differences for electrocardiograms (ECGs) is proposed. ECGs can be used for biometric purposes as stated in [5], where features are extracted by principal component analysis (PCA). The dynamic time warp (DTW) [6] is a distance metric employed for biosignals such as BOLD fMRI signals [7]. The cardinality of large biodatasets can be estimated as in [8]. Arrythmia detection from noisy ECG signals can be done by neural networks [9][10][11], non-linear principal component analysis (PCA) neural networks [12], and zero-pole infinite impulse response (IIR) analysis [13]. Finally, graph-based methods exist for analyzing brain connectivity [14] or protein-to-protein interaction [15] and are supported by graph databases such as Neo4j [16].

Higher order statistics describe the behavior of a random variable beyond the mean value and the variance [17]. Although for a normally distributed random variable the knowledge of these two quantities suffice to completely understand it, this in general is not the case [18][19]. Higher order moments form the theoretical foundation of independent component analysis (ICA) [20], a methodology with numerous applications in signal processing and in bioengineering among others. Higher order cumulants have been applied in the simulation of low level brain activity [21][22][23].

Scheduling policies are of paramount importance in operating systems such as the completely fair scheduler (CFS) found predominantly in Linux systems since the 2.6.23 kernel release [24], in multithreading environments [25][26], in parallel systems [27][28], and in distributed computing frameworks [29]. Also branch prediction strategies found in modern hardware are based on techniques such as neural networks [3] [30]. An early overview can be found in [2]. Given that a VLSI systolic array implemenation for computing higher order moments already exist [31], the proposed policy can be integrated into CPU circuitry.

Finally, since for the build and maintenance of the scheduler some of the GNU *autotools* software tools collection were used, they deserve a brief mention. When a new software package is to be installed, then *configure* executes a special script contained in that package as mandated by the GNU Coding Standards to tailor it for the local system. Then the source code is compiled and with GNU *make*, which controls conditional compilation, library linking, and, in the context of this paper, the handling of some symbolic names. Thus, in a typical Linux system typing at the prompt

```
# configure && make && make install
```

usually sufficies to install a new software package. Finally, *libtool* prepares source code for being part of a static or dynamic library according to essential Linux requirements about the maintenance and interoperability of shared libraries.

III. Higher Order Moments

Higher order statistics are frequently used to obtain a succint description of the behavior of a random variable. Although a normally distributed random variable is fully characterized by its mean and variance, this is not the case for an arbitrary random variable X. Thus, it is desired knolwdge of the higher order moments m_k or of the higher order cumulants c_k. The following discussion is about $m_k \triangleq \mathrm{E}\left[X^k\right]$.

Definition 1: The moment generating function $M_X(s)$ for a random variable X is defined as the conjugate Laplace transform of random variable X, namely

$$M_X(s) \triangleq \mathrm{E}\left[e^{sX}\right], \quad M_X(0) = 1, s \in \mathbb{C} \tag{1}$$

Unlike the characteristic function, the moment generating function may not exist for a given distribution. A prime example is the lognormal distribution where the density $f_{X_l}(\cdot)$ for such a continuous random variable X_l is

$$f_{X_l}(\ln x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}} \tag{2}$$

whose k-th moment is the well defined quantity

$$\mathrm{E}\left[X_l^k\right] = e^{k\mu + \frac{k^2\sigma^2}{2}} \tag{3}$$

When $M_X(s)$ exists, then it holds that

Property 1: The Taylor expansion of $M_X(s)$ is

$$M_X(s) \triangleq \sum_{k=0}^{+\infty} \mathrm{E}\left[X^k\right] \frac{s^k}{k!} \tag{4}$$

and consequently the k-th moment can be computed as

$$\mathrm{E}\left[X^k\right] \;=\; k!\left(\frac{\partial^k M_X(s)}{\partial s}\,|_{\,s=0}\right) \tag{5}$$

The datasets used as benchmarks during the simulation relied on the Poisson and the binomial distributions. A Poisson random variable X_p has the probability mass function

$$\mathrm{prob}\left\{X_p \;=\; k; \lambda_0\right\} \triangleq \frac{\lambda_0^k}{k!} e^{-\lambda_0}, \quad k \in \mathbb{Z}^+, \lambda_0 \in \mathbb{R}^* \tag{6}$$

where its mean value and variance are

$$\mathrm{E}\left[X_p\right] \;=\; \mathrm{Var}\left[X_p\right] \;=\; \lambda_0 \tag{7}$$

and its moment generating function is

$$M_{X_p}(s) \;=\; e^{\lambda_0(e^s-1)} \;\approx\; e^{\lambda_0 s} \tag{8}$$

where the last approximation is due to the MacLaurin expansion of the exponential function

$$e^s \;=\; \sum_{k=0}^{+\infty} \frac{s^k}{k!} \;=\; 1 + s + \frac{s^2}{2} + \frac{s^3}{6} \ldots \;\approx\; 1 + s, |s| \ll 1 \tag{9}$$

The Poisson distribution is unimodal and skewed, in other words it has a unique global maximum and it is nonsymmetric, with a considerable segment of its mass concentrated near the origin. It is used to model a system where there is a large number of short processes mixed with a few big jobs, representing thus a scenario of uneven load.

On the contrary, a system where the majority of jobs is of intermediate length with small and equal fractions of short and long processes is modeled by a binomial random variable. Such a variable X_b has the probability mass function

$$\mathrm{prob}\left\{X_b \;=\; k; n, p_0\right\} \triangleq \binom{n}{p_0} p_0^k (1-p_0)^{n-k} \tag{10}$$

with $n \in \mathbb{Z}^+$ and $p_0 \in (0,1)$. Its mean value and variance are

$$\mathrm{E}\left[X_b\right] \;=\; np_0 \text{ and } \mathrm{Var}\left[X_b\right] \;=\; np_0(1-p_0) \tag{11}$$

while $M_{X_b}(s)$ is

$$M_{X_b}(s) = ((1-p_0) + p_0 e^s)^n \approx e^{-np_0} + np_0 e^{s-(n-1)p_0} \tag{12}$$

where the last relationship is the first order approximation of

$$(\gamma_0 + x)^n \;=\; \sum_{k=0}^{n} \binom{n}{k} \gamma_0^{n-k} x^k \;\approx\; \gamma_0^n + n\gamma_0^{n-1} x \tag{13}$$

in conjunction with the approximation

$$(1-x)^{\gamma_0} \;\approx\; e^{-\gamma_0 x}, \quad |x| \ll 1, \gamma_0 \gg 1 \tag{14}$$

Like the Poisson distribution it is also unimodal but it is not skewed. This property is crucial for establishing that intermediate-sized jobs are the majority in the process pool. This distribution will establish the baseline scenario, since it intuitively correspond to a moderately loaded system.

A common third order metric for assessing the skewness of a given distribution is the *skewness coefficient* [17] defined as

$$\begin{aligned}\kappa \triangleq \mathrm{E}\left[\left(\frac{X - \mathrm{E}\left[X\right]}{\sqrt{\mathrm{Var}\left[X\right]}}\right)^3\right] \;&=\; \frac{\mathrm{E}\left[\left(X - \mathrm{E}\left[X\right]\right)^3\right]}{\mathrm{Var}\left[X\right]^{\frac{3}{2}}} \\ &=\; \frac{\mathrm{E}\left[X^3\right] - 3\mathrm{E}\left[X\right]\mathrm{Var}\left[X\right] - 3\mathrm{E}\left[X\right]}{\mathrm{Var}\left[X\right]^{\frac{3}{2}}} \\ &=\; \frac{\mathrm{E}\left[X^3\right] - 3\mathrm{E}\left[X\right]\left(\mathrm{E}\left[X^2\right] - \left(\mathrm{E}\left[X\right]\right)^2\right) - 3\mathrm{E}\left[X\right]}{\left(\mathrm{E}\left[X^2\right] - \left(\mathrm{E}\left[X\right]\right)^2\right)^{\frac{3}{2}}}\end{aligned}$$

Since the scheduler is agnostic to the process size distribution and κ is time dependent, it has to be estimated. The noncentral and central *natural estimators* of order k, denoted respectively as ν_k and $\bar{\nu}_k$ [17], provide a way for tracking κ. Since

$$\nu_k \triangleq \frac{1}{n-k+1} \sum_{j=0}^{n-1} x^k[j] \;\approx\; \mathrm{E}\left[X^k\right]$$

$$\bar{\nu}_k \triangleq \frac{1}{n-k+1} \sum_{j=0}^{n-1} \left(x[j] - \nu_1\right)^k \approx \mathrm{E}\left[\left(X - \mathrm{E}\left[X\right]\right)^k\right]$$

then it follows that κ is approximated by the ratio

$$\kappa \;\approx\; \frac{\bar{\nu}_3}{\bar{\nu}_2^{\frac{3}{2}}} \;=\; \frac{\nu_3 - 3\nu_1\left(\nu_2 - \nu_1^2\right) - 3\nu_1^3}{\left(\nu_2 - \nu_1^2\right)^{\frac{3}{2}}} \tag{15}$$

It should be noted that the noncentral natural estimator may be suboptimal in terms of estimation error compared to a Bayesian or a maximum likelihood (ML) estimator [17], but it can be easily recursively computed as new jobs arrive as

$$\nu_k[n+1] \;=\; \left(\frac{n}{n+1}\right)\nu_k[n] + \left(\frac{1}{n+1}\right)x^k[n] \tag{16}$$

where $\nu_k[n]$ is the natural estimator of order k for n processes. Recall that $\nu_k[n]$ pertains to job set $\{x[j]\}_{j=0}^{n-1}$ while $\nu_k[k]$ to $\{x[j]\}_{j=0}^{n-1} \cup \{x[n]\}$. Since there is no lightweight way to express the central natural estimator in recursive terms, (15) was cast in terms of natural estimators based on the fact that

$$\mathrm{Var}\left[X\right] \;=\; \mathrm{E}\left[X^2\right] - \left(\mathrm{E}\left[X\right]\right)^2 \;\approx\; \nu_2 - \nu_1^2 \tag{17}$$

In the case of unimodal distributions κ might, under some assumptions, give an indication of the distribution shape. Specifically, when κ equals zero, then the underlying distribution is either symmetric or asymmetric but with equal mass on both sides of the mean. In other words, in this case the mean is also the median of the distribution. When κ is positive, then more mass is placed on the right tail of the distribution, while a negative κ denotes a mass shift to the left tail. A binomially distributed random variable has κ equal to zero, while for a Poisson random variable κ is $1/\lambda_0$.

A practical problem which arose during the dataset creation was the generation of random numbers. Instead of using

specialized software such as MATLAB or R, two established mathematical approaches were implemented in C in the interests of keeping the number of software tools low. In order to create a sequence of Poisson distributed numbers, the *inverse transform* method, which relies on uniformly distributed numbers over the unitary interval [17], was implemented as shown in algorithm 1 using the follwing code snippet

```
#include <stdlib.h>
#include <time.h>
#include <math.h>
...
time_t t0; double u;
t0 = time((time_t)NULL);
srand((unsigned int)t0);
u = fmod(rand(), 1.0 + (double)RAND_MAX);
```

The rand() function generates unformly distributed integers from 0 to the implementation defined constant RAND_MAX, which is guaranteed to be at least $2^{16} - 1$ in ANSI C and subsequent standards. The casting to **double** and the addition of an explicitly double literal prevent a denominator overflow as in many implementations RAND_MAX is the maximum **int** or **long**. srand() does not return a value and is used only to initialize the pseudorandom number generator of the standard C library. The expensive double precision operations are not part of the simulation and were only used once for the dataset generation before the actual simulation. Also, the fmod() function is necessary as the C modulus operator $\%$ works only with integer operands.

Algorithm 1 *Inverse transform* method for Poisson r.v. [17]

Require: $U = \{u_k\}$ uniformly distributed in $(0, 1)$; λ_0
Ensure: $P = \{p_k\}$ has Poisson distributed numbers
1: **for** $k \leftarrow 1$ **to** $|U|$ **do**
2: $j \leftarrow 0$ **and** $p \leftarrow e^{-\lambda_0}$ **and** $F \leftarrow p$
3: **while** $u_k > F$ **do**
4: $p \leftarrow \frac{p\lambda_0}{j+1}$ **and** $F \leftarrow F + p$ **and** $j \leftarrow j + 1$
5: **end while**
6: $p_k \leftarrow j$ **and** $P \leftarrow P \cup \{p_k\}$
7: **end for**
8: **return** P

The binomially distributed numbers were generated in a straightforward way based on the fact that X_b is the sum of n independent indicator or Bernoulli random variables. Therefore, algorithm 2 was used in this case.

Algorithm 2 Binomially distributed number generation

Require: Parameters n and p_0; desired cardinality $|B|^*$
Ensure: B has n binomially distributed numbers as in (10)
1: $B \leftarrow \emptyset$ **and** $|B| \leftarrow 0$
2: **while** $|B| < |B|^*$ **do**
3: get independent indicator r.vs $\{I_k\}_{k=1}^n$, each with p_0
4: $b \leftarrow \sum_{k=1}^n I_k$ **and** $B \leftarrow B \cup \{b\}$ **and** $|B| \leftarrow |B| + 1$
5: **end while**
6: **return** B

IV. Proposed Scheduling Policy

Scheduling policies lie at the heart of virtually all production and development systems ranging from personal computers and workstations to mainframes to distributed processing systems. They play an important role since they directly affect average process completion time, which is of essence in case of many interconnected systems, as well as the perceived system performance, an integral part of user experience. Besides process profile, the mission requirement of a system is also a factor in scheduling policy selection.

Some basic common policies include:

- First in, first out (FIFO): Processes are executed in the order they arrive. Although it is straightforward to implement and maintain, the average execution time under FIFO depends heavily on the process arrival sequence. The latter is in general undesired, as it may lead to slow, fluctuating, or otherwise unacceptable system performance for specific process arrival sequences.
- Shortest job first (SJF): Processes are shorted in ascending estimated execution time and then are executed in that order. Average execution time tends to gradually increase, while the insertion of new short processes in the system may result in long delays for bigger jobs.
- Round robin (RR): Each process is sequentially given a time slot. Its major drawback is frequent context switching, although it leads to acceptable performance.

In terms of its dominant resource a process can be categorized as follows:

- CPU bound: The process type is computationally intensive requiring frequent access to the CPUs or to the GPUs. Although a CPU bound application need noy necessarily be data intensive, it is rapidly becoming the case. Typically have the highest priority in hierarchical scheduling policies.
- I/O bound: This process type is usually interactive in nature or handles large data volumes. Such a process tends to be blocked waiting for user response or reading disk files or generating traffic along the memory hierarchy. An I/O bound process is more likely than a CPU bound to be data intensive.
- Memory bound: A process of this type also utilizes heavily the memory hierarchy but, unlike an I/O bound process, both its execution time and messaging burden are inversely proportional to total memory capacity. It is primarily used to describe big data applications relying on sublinear or random sampling algorithms.
- Network bound: This is an emerging category intended to serve as an abstraction of streaming big data applications. Unlike a memory bound process, the window length does not necessarily affect computation time but it is closely related to result quality. A process of this type relies on network resources, which are slower than CPUs and

GPUs but faster than typical memory components [32].

At this point a crucial distinction should be made. Any job x has a length $J(x)$, so the $\mathrm{E}[J(x)]$ and $\mathrm{Var}[J(x)]$ are well defined, whether they are deterministic or stochastic. However, when processes are scheduled, then the mean $\mathrm{E}[X]$ and the avarage $\mathrm{Var}[X]$ of the actual computation time each process requires under workload ρ_0 are also of interest. In general, $\mathrm{E}[J(x)] \neq \mathrm{E}[X]$ and $\mathrm{Var}[J(x)] \neq \mathrm{Var}[X]$ because of a multitude of reasons such as scheduling overhead, context switching, I/O requests, cache misses, memory transfers, and high latency disk reads [33]. The following criteria assess scheduling quality in terms of these quantities.

Criterion 1: $\mathrm{E}[X]$ should be a linear function of $\mathrm{E}[J(x)]$

$$\mathrm{E}[X] = \theta_1 \mathrm{E}[J(x)] + \theta_0 \tag{18}$$

This criterion provides a service guarantee regarding $\mathrm{E}[X]$. Parameters θ_1 and θ_0 depend on ρ_0 and may not always exist.

In scenaria when systems are pipelined or when a smooth job flow is required either for technological, financial, or user experience reasons metrics regarding $\mathrm{Var}[X]$ are required.

Criterion 2: The difference of the job execution time variance $\mathrm{Var}[X]$ from $\mathrm{Var}[J(x)]$ normalized by $\mathrm{Var}[J(x)]$ should not exceed a constant, namely

$$\frac{|\mathrm{Var}[X] - \mathrm{Var}[J(x)]|}{\mathrm{Var}[J(x)]} \leq \theta_2 \tag{19}$$

Criterion 2 places stricter constraints on the scheduling policy but ensures both a bounded process execution time and a smoother job flow. Here θ_2 depends on ρ.

Criterion 3: Given that w_0 processes are scheduled, there should be at least $\theta_3 = \Omega(w_0)$ time steps before a switch.

Since algorithm 3 relies on scheduling switching, it makes sense to investigate how frequently policy changes occur since they incur an non-negligible cost. Criterion 3 outlines a possible acceptable frequency which gives to each process at least a constant amount of timesteps before an estimation of its execution time is formed.

Higher Order Policy (HOP), the proposed scheduler aimed mostly at CPU bound processes, is outlined in algorithm 3. It is based on the observation that SJF is on the average more apppropriate when the process length distribution is unbalanced as it yields a more stable job flow. As shorter processes are completed and moved out of the job pool, then fewer and larger jobs remain. As a result, the process length distribution tends to be more balanced and the overall job size variance decreases, a scenario in which RR results in smoother job flow. In case the process size becomes again unbalanced, perhaps as new jobs arrive or because of local fluctuations at the job length distribution, switching returns to SJF mode.

HOP requires besides the process set S four operational parameters, namely w_0, β_0, τ_0, and L. Scheduling stability is primarily ensured by L and w_0. Specifially, L is the maximum number of scheduled jobs at any given moment and it is a safeguard against thrashing. Although other sophisticated techniques exist, the crude mechanism of keeping track of the number of scheduled jobs suffices in many applications. The full study of thrashing phenomena are outside the scope of this paper.

Stability is additionally ensured by w_0, the number of most recent past estimates used for computing κ. A tradeoff was sought for determining w_0, as a large value would smooth job lengths but the smoothed average might contain samples from an older process length distribution. On the contrary, a small value of w_0 will always contain with high probability samples from the current distribution but possibly at the expense of reduced estimation error. The time window within which no policy switch is allowed is also controlled by w_0 and equals $\beta_0 w_0$. According to criterion 3, many functions of w_0 would be admissible. However, a simple one was chosen which is directly dependent on w_0.

Finally, τ_0 is a numerical threshold which allows some tolerence in declaring that κ is zero. Its inclusion was necessary not only because of numerical reasons, namely floating point roundoff errors, but also because of algorithmic ones: Since κ is approximated by an estimator known to be error prone and, moreover, the estimator itself is estimated over a time window, it makes sense to assume that κ might not be zero but close to zero in the presence of the Poisson distribution.

Algorithm 3 Proposed scheduling policy (HOP)

Require: Process set S; thresholds w_0, β_0, τ_0, L
Ensure: Conforms to quality and performance requirements

```
while job set S is non-empty do
    if a current job s* has terminated then
        S ← S \ {s*}
    end if
    if a new job s' is waiting and |S| < L then
        S ← S ∩ {s'}
    end if
    if |S| < w_0 then
        schedule with RR
    else
        update κ as in (15) with the most recent w_0 values
        if no policy change during past β_0 w_0 steps then
            if |κ| ≤ τ_0 and policy is RR then
                switch to SJF
            else
                if |κ| > τ_0 and policy is SJF then
                    switch to RR
                end if
            end if
        end if
    end if
end while
return
```

Some general notes about the simulation in the following two sections are in order. There were no dependencies among processes and the single most important resource for each job was memory. Moreover, the total memory requirements were less than the physically available RAM. The combined effect

of these factors were the absence of deadlocks, extensive cache misses, and excessive disk utilization.

Processes were chosen to be scheduled with the lightweight *execve* function declared in the header <unistd.h> and which directly opens executable files. The source code was compiled with the standard *gcc* version of Ubuntu 16.04.01 *Xenial Xerus* LTS to an executable file which had the ELF file format and supported 64-bit address spaces. Also, the C code was maintained with GNU *make*, the installation script was automatically prepared with *configure* and a shared library version where the schedulers as well as the auxiliary pre-sudorandom number generators are available as library calls was generated with *libtool*. The development platform was an Ubuntu 16.04.01 LTS workstation with 16 GB of physical memory and an Intel Core i7 processor with 4×256 KB L2 cache and 8MB L3 cache.

Finally, although the proposed scheduling policy simulates an operating system function, it was compiled in *hosted* mode which is typical for higher level C applications. Therefore, compared to the *free-standing* compilation mode for low level code, there were unusually many programming facilities available, such as the standard C I/O library and higher level system calls. The only such facility which was actually used were functions from the mathematical library. The primary data types and structures such as **union** and **struct** are part of the language proper and, hence, are available in both modes.

V. Application: Z-score Computation

Data vectors often have to undergo some normalization before comparison, especially if the distance metric to be applied is sensitive to various translations. A normalization common in biomedical signal processing is Z-score [7] which given the sample mean and variance of the data vector $\mathbf{x}$

$$\begin{aligned}\hat{\mu}_0 &\triangleq \frac{1}{n}\sum_{k=0}^{n-1}\mathbf{x}[k] \;=\; \nu_1 \\ \hat{\sigma}_0 &\triangleq \sqrt{\frac{1}{n-1}\sum_{k=0}^{n-1}\left(\mathbf{x}[k]-\hat{\mu}_0\right)^2} \;=\; \sqrt{\nu_2}\end{aligned} \tag{20}$$

generates the normalized version $\mathbf{z}$ for $0 \le k \le n-1$

$$\mathbf{z}[k] \triangleq \begin{cases}\frac{1}{\hat{\sigma}_0}(\mathbf{x}[k]-\hat{\mu}_0), & \hat{\sigma}_0 \ge \tau_0 \\ \hat{\mu}_0, & \hat{\sigma}_0 < \tau_0\end{cases} \tag{21}$$

This double normalization, in terms of both amplitude and variance, is pivotal for certain pattern discovery algorithms and distance metrics. Consider for instance a vector $\mathbf{x} \in \mathbb{R}^n$ representing a discrete signal and a scaled version of itself $\delta_0\mathbf{x}$. If the useful information is codified in the waveform represented by $\mathbf{x}$ and not in the scaling parameter δ_0, as in FSK modulation, then the Euclidean distance will fail to discover a perfect match. In case the variance is below a threshold, which practically means that $\mathbf{x}$ is almost constant, no division takes place and the signal is simply replaced by $\hat{\mu}_0$ as a numerical safeguard preventing a very small denominator. The complexity of algorithm 4 is $\Theta(n)$ and, once $\hat{\mu}_0$ and $\hat{\sigma}_0$ are computed, the Z-score of each $\mathbf{x}[k]$ can be computed at constant time and memory, making it a lightweight procedure.

Algorithm 4 Z-score computation (basic implementation)

Require: Data $\mathbf{x}$; threshold τ_0
Ensure: $\mathbf{z}$ is a doubly normalized version of $\mathbf{x}$

$\hat{\mu}_0 \leftarrow 0$ **and** $\hat{\sigma}_0 \leftarrow 0$
for $j \leftarrow 0$ **to** $n-1$ **do**
  $\hat{\mu}_0 \leftarrow \hat{\mu}_0 + \mathbf{x}[j]$
end for
$\hat{\mu}_0 \leftarrow \frac{\hat{\mu}_0}{n}$
for $j \leftarrow 0$ **to** $n-1$ **do**
  $\hat{\sigma}_0 \leftarrow \hat{\sigma}_0 + (\mathbf{x}[j]-\hat{\mu}_0)^2$
end for
$\hat{\sigma}_0 \leftarrow \sqrt{\frac{1}{n-1}\hat{\sigma}_0}$
if $\hat{\sigma}_0 \le \tau_0$ **then**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $\mathbf{z}[j] \leftarrow \hat{\mu}_0$
  end for
else
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $\mathbf{z}[j] \leftarrow \frac{1}{\hat{\sigma}_0}(\mathbf{x}[j]-\hat{\mu}_0)$
  end for
end if
return $\mathbf{z}$

In order to simulate the workload of a biosignal processing system, the publically available MIT-BIH dataset was used to generate the benchmark dataset. Specifically, the *series 1* waveform with 1800 samples was designated as the generator signal. Then, 2000 continuous segments, 1000 for each distribution, were extracted always starting from the first sample. The length of each such segment was chosen according to

- either a binomial distribution with $n = 400$ and $p_0 = 0.4$. $\mathrm{E}[J(x_b)] = 421$ and $\mathrm{Var}[J(x_b)] = 72$.
- or a Poisson distribution with $\lambda_0 = 0.5$ added to the first 300 samples to ensure bigger segments. $\mathrm{E}[J(x_p)] = 414$ and $\mathrm{Var}[J(x_p)] = 16$.

The data were copied with a C program to the C source code of the application as static data in order to avoid costly file reads. In this context the workload ρ_0 means that at any given at most ρ_0 percent of the total processes are being scheduled.

Subsequently the average $\bar{\theta}_0$, $\bar{\theta}_1$, $\bar{\theta}_2$, and $\bar{\theta}_3$ were computed. $\bar{\theta}_0$ and $\bar{\theta}_1$ were approximated in the least squares sense. By design, only HOP results in policy switching.

As shown in tables II and III, the unbalanced nature of the Poisson distribution is reflected to the almost double size of policy switches compared to the ones required on average for the balanced binomial dataset. There are policy changes in the latter as well, but their number suggests they have occured to counter some temporary load imbalance.

Regarding the variance difference, as reflected in $\bar{\theta}_2$, its behavior is similar in both datasets. That is, FIFO leads to the biggest such difference denoting a low QoS. This can be attributed to the lack of control over the job sequence. HOP and RR are almost equivalent, while SJF is in-between.

TABLE II
Z-SCORE COMPUTATION TIME (BINOMIAL, $\rho_0 = 0.2$).

Policy	θ_1	θ_0	θ_2	θ_3
FIFO	1.1902	0.9999	1.4916	-
RR	1.2432	1.8223	1.1092	-
SJF	1.2175	1.7082	1.3217	-
HOP	1.1899	2.1841	1.0113	12.4309

Finally, HOP achieves a better relationship with a lower $\bar{\theta}_1$ between $\mathrm{E}[J(x)]$ and $\mathrm{E}[X]$ in both cases, where now RR has the worst average performance because of frequent context switching. FIFO and RR have comparable performance.

TABLE III
Z-SCORE COMPUTATION TIME (POISSON, $\rho_0 = 0.2$).

Policy	θ_1	θ_0	θ_2	θ_3
FIFO	1.3844	1.2100	1.7702	-
RR	1.5182	1.7721	1.1991	-
SJF	1.3157	1.4442	1.5180	-
HOP	1.2995	2.5224	1.2021	26.1131

VI. APPLICATION: GÖRTZEL ALGORITHM COMPUTATION

The Fourier transform of a biosignal plays an important role in many cases, for instance in the analysis of electrocardiograms (ECGs). Since the human heart has been long evolved to operate in a clockwork periodic way, any signs of malfunction such as tachycardia or bradycardia might show easier in a spectral analysis of an ECG. Although Fast Fourier Transform is in every aspect an efficient algorithm and has been implementated on numerous platforms, there are instances where selective frequency monitor is desired. When b, the number of frequencies to be moniotred, is less than $\mathrm{O}(\log n)$, where n is the dataset size, then the Görtzel algorithm, a methodology based on the principles of multirate signal processing might be an appealing choice, provided that n is of moderate size.

The Görtzel algorithm is an IIR filter with transfer function

$$H(z) = \frac{1 - \omega_k z^{-1}}{1 - 2\cos\left(\frac{2k\pi}{n}\right) z^{-1} + z^{-2}}, \quad \omega_k = e^{i\frac{2k\pi}{n}} \tag{22}$$

which corresponds to the recurrence for $0 \leq j \leq n-1$

$$\mathbf{u}[j] = \mathbf{x}[j] - 2\cos\left(\frac{2k\pi}{n}\right)\mathbf{u}[j-1] - \mathbf{u}[j-2]$$

with the elements of the auxiliary data vector $\mathbf{u}$ are

$$\mathbf{u}[0] = \mathbf{x}[0] \text{ and } \mathbf{u}[1] = \mathbf{x}[1] - 2\cos\left(\frac{2k\pi}{n}\right)\mathbf{x}[0] \tag{23}$$

and the final value in $\mathbf{f}[k-1]$ is

$$\mathbf{f}[k-1] = \mathbf{u}[n-1] - \omega_k \mathbf{u}[n-2] \tag{24}$$

The basic form is outlined in algorithm 5. Its complexity is $\mathrm{O}(bn)$ which can be quadratic at worst when $b = \mathrm{O}(n)$.

Algorithm 5 Görtzel algorithm (basic implementation)

Require: Data $\mathbf{x}$; frequency set $\Omega \triangleq \{\omega_k\}_{k=1}^{n}$
Ensure: $\mathbf{f}[k-1]$ contains the Fourier coefficient at ω_k
1: **for all** $\omega_k \in \Omega$ **do**
2: $\quad \mathbf{u}[0] \leftarrow \mathbf{x}[0]$ **and** $\mathbf{u}[1] \leftarrow \mathbf{x}[1] - 2\cos\left(\frac{2k\pi}{n}\right)\mathbf{x}[0]$
3: $\quad$ **for** $j \leftarrow 2$ **to** $n-1$ **do**
4: $\quad\quad \mathbf{u}[j] \leftarrow \mathbf{x}[j] - 2\cos\left(\frac{2k\pi}{n}\right)\mathbf{u}[j-1] - \mathbf{u}[j-2]$
5: $\quad$ **end for**
6: $\quad \mathbf{f}[k-1] \leftarrow \mathbf{u}[n-1] - \omega_k \mathbf{u}[n-2]$
7: **end for**
8: **return** $\mathbf{f}$

The same methodology as in the previous section was followed and the results are shown in tables IV to VII. In general, the behavior of the four scheduling algorithms is the same with that of the previous section. However, becuase of the increased complexity of Görtzel filtering, there are sharper differences in performance. FIFO and SJF now achieve the best linear relationship with HOP being the runner up but with better variance than the two. Therefore, HOP maintains a better mean-variance tradeoff compared to its competitors. Also, the number of scheduling switches is again roughly the half for the balanced dataset compared to the unbalanced. Finally, bigger b results in more demanding computations.

TABLE IV
GÖRTZEL COMPUTATION TIME (BINOMIAL, $b = 5$, $\rho_0 = 0.2$).

Policy	θ_1	θ_0	θ_2	θ_3
FIFO	1.2331	1.5789	2.7709	-
RR	1.9021	1.9336	1.8733	-
SJF	1.5513	1.4325	2.1133	-
HOP	1.7003	1.3565	1.8236	17.0022

TABLE V
GÖRTZEL COMPUTATION TIME (POISSON, $b = 5$, $\rho_0 = 0.2$).

Policy	$\hat{\theta}_1$	$\hat{\theta}_0$	$\hat{\theta}_2$	$\hat{\theta}_3$
FIFO	1.2846	1.6973	3.3429	-
RR	1.9734	2.0031	2.0888	-
SJF	1.4520	1.6643	2.5517	-
HOP	1.5009	1.1135	1.9099	31.0251

TABLE VI
GÖRTZEL COMPUTATION TIME (BINOMIAL, $b = 100$, $\rho_0 = 0.2$).

Policy	$\hat{\theta}_1$	$\hat{\theta}_0$	$\hat{\theta}_2$	$\hat{\theta}_3$
FIFO	1.2703	1.8856	3.8592	-
RR	2.0798	1.9964	2.1831	-
SJF	1.4118	1.2241	2.6113	-
HOP	1.6733	2.3354	2.1105	14.8875

TABLE VII
GÖRTZEL COMPUTATION TIME (POISSON, $b = 100$, $\rho_0 = 0.2$).

Policy	$\hat{\theta}_1$	$\hat{\theta}_0$	$\hat{\theta}_2$	$\hat{\theta}_3$
FIFO	1.4283	1.7902	4.0003	-
RR	2.0934	1.7773	2.5802	-
SJF	1.5633	1.9425	2.7718	-
HOP	1.5172	1.8809	2.2993	25.9163

VII. CONCLUSIONS AND FUTURE WORK

This paper proposes HOP, an adaptive scheduling policy at the operating system level which relies on switching between SJF and RR depending on the estimation of the skewness coefficient of the job size distribution. Simulations with two biomedial analytics applied each to two datasets derived from the MIT-BIH standard dataset indicates that HOP offers improved load handling and reduced execution time variance.

Future directions include a better estimator like the median filter, the investigation of the relationship between the window size and the estimation error, and an extension to Hadoop.

ACKNOWLEDGEMENTS

This work was supported by the EU Horizon 2020 research and innovation programme under grant agreement No 690140.

REFERENCES

[1] A. Rasooli and D. G. Down, "An adaptive scheduling algorithm for dynamic heterogeneous hadoop systems," in Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research. IBM, 2011, pp. 30–44.

[2] J. E. Smith, "A study of branch prediction strategies," in Proceedings of the 8th annual symposium on Computer Architecture. IEEE Computer Society Press, 1981, pp. 135–148.

[3] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in The Seventh International Symposium on High-Performance Computer Architecture, 2001. IEEE, 2001, pp. 197–206.

[4] G. Drakopoulos and V. Megalooikonomou, "Regularizing large biosignals with finite differences," in Proceedings of the 6th International Conference of Information, Intelligence, Systems, and Applications, ser. IISA 2016. IEEE, July 2016.

[5] L. Biel, O. Pettersson, L. Philipson, and P. Wide, "ECG analysis: A new approach in human identification," IEEE Transactions on Instrumentation and Measurement, vol. 50, no. 3, 2001, pp. 808–812.

[6] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," in IEEE 23rd International Conference on Data Engineering, ser. 2007. IEEE, 2007, pp. 1046–1055.

[7] F. Tagkalakis, A. Papagiannaki, G. Drakopoulos, and V. Megalooikonomou, "Augmenting fMRI-generated brain connectivity with temporal information," in Proceedings of the 6th International Conference of Information, Intelligence, Systems, and Applications, ser. IISA 2016. IEEE, July 2016.

[8] G. Drakopoulos, S. Kontopoulos, and C. Makris, "Eventually consistent cardinality estimation with applications in biodata mining," in Proceedings of the 31st Symposium on Applied Computing, ser. SAC 2016. ACM, April 2016.

[9] R. Silipo and C. Marchesi, "Artificial neural networks for automatic ECG analysis," IEEE Transactions on Signal Processing, vol. 46, no. 5, 1998, pp. 1417–1425.

[10] N. V. Thakor and Y.-S. Zhu, "Applications of adaptive filtering to ECG analysis: Noise cancellation and arrhythmia detection," IEEE Transactions on Biomedical Engineering, vol. 38, no. 8, 1991, pp. 785–794.

[11] J. P. Abenstein and W. J. Tompkins, "A new data-reduction algorithm for real-time ECG analysis," IEEE Transactions on Biomedical Engineering, no. 1, 1982, pp. 43–48.

[12] T. Stamkopoulos, K. Diamantaras, N. Maglaveras, and M. Strintzis, "ECG analysis using nonlinear PCA neural networks for ischemia detection," IEEE Transactions on Signal Processing, vol. 46, no. 11, 1998, pp. 3058–3067.

[13] I. Murthy and G. Prasad, "Analysis of ECG from pole-zero models," IEEE Transactions on Biomedical Engineering, vol. 39, no. 7, 1992, pp. 741–751.

[14] O. Sporns, "Graph-theoretical analysis of brain networks," in Brain Mapping: An Encyclopedic Reference, A. W. Toga, Ed. Academic Press: Elsevier, December 2015, vol. 1, pp. 629–633.

[15] D. Hoksza et al., "Using Neo4j for mining protein graphs: A case study," in 26th international conference on Database and Expert Systems Applications (DEXA). IEEE, 2015, pp. 230–234.

[16] G. Drakopoulos, A. Baroutiadi, and V. Megalooikonomou, "Higher order graph centrality measures for Neo4j," in Proceedings of the 6th International Conference of Information, Intelligence, Systems, and Applications, ser. IISA 2015. IEEE, July 2015.

[17] A. Papoulis and S. U. Pillai, Probability, random variables, and stochastic processes. McGraw-Hill Education, 2002.

[18] J.-F. Cardoso, "Source separation using higher order moments," in International Conference on Acoustics, Speech, and Signal Processing, ser. ICASSP-89. IEEE, 1989, pp. 2109–2112.

[19] R. Pan and C. L. Nikias, "The complex cepstrum of higher order cumulants and nonminimum phase system identification," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 36, no. 2, 1988, pp. 186–205.

[20] P. Comon, "Independent component analysis, a new concept?" Signal Processing, vol. 36, no. 3, 1994, pp. 287–314.

[21] B. Staude, S. Rotter, and S. Grün, "CuBIC: Cumulant based inference of higher-order correlations in massively parallel spike trains," Journal of Computational Neuroscience, vol. 29, no. 1-2, 2010, pp. 327–350.

[22] B. Staude and S. Rotter, "Higher-order correlations in non-stationary parallel spike trains: Statistical modeling and inference," BMC Neuroscience, vol. 10, no. Suppl 1, 2009, p. P108.

[23] H. Shimazaki, S.-i. Amari, E. N. Brown, and S. Grün, "State-space analysis of time-varying higher-order spike correlation for multiple neural spike train data," PLoS Comput Biol, vol. 8, no. 3, 2012, p. e1002385.

[24] C. Singh Pabla, "Completely fair scheduler," Linux Journal, vol. 184, no. 1, August 2009.

[25] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," Journal of the ACM, vol. 46, no. 5, 1999, pp. 720–748.

[26] A. Snavely, D. M. Tullsen, and G. Voelker, "Symbiotic job scheduling with priorities for a simultaneous multithreading processor," in ACM SIGMETRICS Performance Evaluation Review, vol. 30, no. 1. ACM, 2002, pp. 66–76.

[27] A.-H. Haritatos, G. Goumas, N. Anastopoulos, K. Nikas, K. Kourtis, and N. Koziris, "LCA: A memory link and cache-aware co-scheduling approach for CMPs," in Proceedings of the 23rd International Conference on Parallel Architectures and Compilation, ser. PACT '14. New York, NY, USA: ACM, 2014, pp. 469–470. [Online]. Available: http://doi.acm.org/10.1145/2628071.2628123

[28] T. A. Wagner, V. Maverick, S. L. Graham, and M. A. Harrison, "Accurate static estimators for program optimization," SIGPLAN Not., vol. 29, no. 6, June 1994, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/773473.178251

[29] T. Sandholm and K. Lai, "Dynamic proportional share scheduling in Hadoop," in Workshop on Job Scheduling Strategies for Parallel Processing. Springer, 2010, pp. 110–131.

[30] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in Proceedings of the 24th annual international symposium on Microarchitecture. ACM, 1991, pp. 51–61.

[31] E. Manolakos and H. Stellakis, "Adaptive computation of higher order moments and its systolic realization," International Journal of Adaptive Control Signal Process, vol. 10, 1996, pp. 283–302.

[32] J. Dean, "Designs, lessons, and advice from building large distributed systems," Keynote from LADIS'09, 2009, p. 1.

[33] A. Wierman and M. Harchol-Balter, "Classifying scheduling policies with respect to higher moments of conditional response time," ACM SIGMETRICS Performance Evaluation Review, vol. 33, no. 1, 2005, pp. 229–240.